Approximate Convex Optimization by Online Game Playing

Elad Hazan *

IBM Almaden Research Center

hazan@us.ibm.com

Abstract

Lagrangian relaxation and approximate optimization algorithms have received much attention in the last two decades. Typically, the running time of these methods to obtain a ε approximate solution is proportional to $\frac{1}{\varepsilon^2}$. Recently, Bienstock and Iyengar, following Nesterov, gave an algorithm for fractional packing linear programs which runs in $\frac{1}{\varepsilon}$ iterations. The latter algorithm requires to solve a convex quadratic program every iteration - an optimization subroutine which dominates the theoretical running time.

We give an algorithm for convex programs with strictly convex constraints which runs in time proportional to $\frac{1}{\varepsilon}$. The algorithm does **not** require to solve any quadratic program, but uses gradient steps and elementary operations only. Problems which have strictly convex constraints include maximum entropy frequency estimation, portfolio optimization with loss risk constraints, and various computational problems in signal processing.

As a side product, we also obtain a simpler version of Bienstock and Iyengar's result for general linear programming, with similar running time.

We derive these algorithms using a new framework for deriving convex optimization algorithms from online game playing algorithms, which may be of independent interest.

1 Introduction

The design of efficient approximation algorithms for certain convex and linear programs has received much attention in the previous two decades. Since interior point methods and other polynomial time algorithm are often too slow in practice [Bie01], researchers have tried to design approximation algorithms. Shahrokhi and Matula [SM90] developed the first approximation algorithm for the maximum concurrent flow problem. Their result spurred a great deal of research, which generalized the techniques to broader classes of problems (linear programming, semi-definite programming, packing and covering convex programs) and improved the running time [LSM+91, KPST94, PST91, GK94, GK98, Fle00, GK95, KL96, AHK05b].

In this paper we consider approximations to more general convex programs. The convex feasibility problem we consider is of the following form (the optimization version can be reduced to this feasibility problem by binary search),

$$f_j(x) \leq 0 \quad \forall j \in [m] \tag{1}$$
$$x \in \mathbb{S}_n$$

Where $\{f_j, j \in [m]\}$ is a (possibly infinite) set of convex constraints and $\mathbb{S}_n = \{x \in \mathbb{R}^n, \sum_i x_i = 1, x_i \geq 0\}$ is the unit simplex. Our algorithm work almost without change if the simplex is replaced

by other simple convex bodies such as the ball or hypercube. The more general version, where $\mathbb{S}_n$ is replaced by an arbitrary convex set in Euclidian space, can also be handled at the expense of slower running time (see section 3.1).

We say that an algorithm gives an ε-approximate solution to the above program if it returns $x \in \mathcal{P}$ such that $\forall j \in [m] . f_j(x) \le \varepsilon$, or returns proof that the program is infeasible. Hence, in this paper we consider an additive notion of approximation. A multiplicative ε-approximation is a $x \in \mathcal{P}$ such that $\forall j \in [m] . f_j(x) \le \lambda^*(1+\varepsilon)$ where $\lambda^* = \min_{x \in \mathcal{P}} \max_{i \in [m]} f_i(x)$. There are standard reductions which convert an additive approximation into a multiplicative approximation. Both of these reductions are orthogonal to our results and can be applied to our algorithms. The first is based on simple scaling, and is standard in previous work (see [PST91, You95, AHK05a]) and increases the running time by a factor of $\frac{1}{\lambda^*}$. For the special case fractional packing and covering problems, there is a different reduction based on binary search which increases the running time only by a poly-logarithmic factor [BI04, Nes04].

A common feature to all of the prior algorithms is that they can be viewed, sometimes implicitly, as Frank-Wolfe [FW56] algorithms, in that they iterate by solving an optimization problems over $\mathbb{S}_n$ (more generally over the underlying convex set), and take convex combinations of iterates. The optimization problem that is iteratively solved is of the following form.

$$\forall p \in \mathbb{S}_m \ . \ \text{OPTIMIZATION ORACLE (p)} \triangleq \begin{cases} x \in \mathbb{S}_n \ \text{s.t} \ \sum_j p_j f_j(x) \le 0 & \text{if exists such } x \\ \\ FAIL & \text{otherwise} \end{cases}$$

It is possible to extend the methods of PST [PST91] and others to problems such as (1) (see [Jan06, Kha04]) and obtain the following theorem. Henceforth ω stands for the *width* of the instance — a measure of the size of the instance numbers — defined as $\omega = \max_{j \in [m]} \max_{x \in \mathbb{S}_n} f_i(x) - \min_{j \in [m]} \min_{x \in \mathbb{S}_n} f_i(x)$.

Theorem 1 (previous work). *There exists an algorithm that for any $\varepsilon > 0$, returns a ε-approximation solution to mathematical program (1). The algorithm makes at most $\tilde{O}(\frac{\omega^2}{\varepsilon^2})$ calls to* OPTIMIZATION ORACLE, *and requires $O(m)$ time between successive oracle calls.*

Remark 1: Much previous work focuses on reducing the dependance of the running time on the width. Linear dependence on ω was achieved for special cases such as *packing and covering problems* (see [You95]). For covering and packing problems the dependence on the width can be removed completely, albeit introducing another n factor into the running time [Jan06]. These results are orthogonal to ours, and it is possible that the ideas can be combined.

Remark 2: In case the constraint functions are linear, OPTIMIZATION ORACLE can be implemented in time $O(mn)$. Otherwise, the oracle reduces to optimization of a convex non-linear function over a convex set.

Klein and Young [KY99] proved an $\Omega(\varepsilon^{-2})$ lower bound for Frank-Wolfe type algorithms for covering and packing linear programs under appropriate conditions. This bound applies to all prior lagrangian relaxation algorithms till the recent result of Bienstock and Iyengar [BI04]. They give an algorithm for solving packing and covering linear programs in time linear in $\frac{1}{\varepsilon}$, proving

Theorem 2 ([BI04]). *There exists an algorithm that for any $\varepsilon > 0$, returns a ε-approximation solution to packing or covering linear programs with m constraints. The algorithm makes at most $\tilde{O}(\frac{n}{\varepsilon})$ iterations. Each iteration requires solving a convex separable quadratic program. The algorithm requires $O(mn)$ time between successive oracle calls.*

Their algorithm has a non-combinatorial component, viz., solving convex separable quadratic programs. To solve these convex programs one can use interior point methods, which have large polynomial running time largely dominating the entire running time of the algorithm. The [BI04] algorithm is based on previous algorithms by Nesterov [Nes04] for special cases of linear and conic programming. Nesterov's algorithm pre-computes a quadratic program, which also dominates the running time of his algorithm.

1.1 Our results

We give a simple approximation algorithms for convex programs whose running time is linear in $\frac{1}{\varepsilon}$. The algorithms requires only gradient computations and combinatorial operations (or a separation oracle more generally), and does not need to solve quadratic programs.

The $\Omega(\varepsilon^{-2})$ lower bound of Klein and Young is circumvented by using the strict convexity of the constraints. The constraint functions are said to be strictly convex if there exists a positive real number $H > 0$ such that $\min_{j \in [m]} \min_{x \in \mathcal{P}} \nabla^2 f_j(x) \succeq H \cdot I$ [1]. In other words, the Hessian of the constraint function is positive definite (as opposed to positive semi-definite) with smallest eigenvalue at least $H > 0$.

Our running time bounds depend on the gradients of the constraint functions as well. Let $G = \max_{j \in [m]} \max_{x \in \mathcal{P}} \|\nabla f_j(x)\|_2$ be an upper bound on the norm of the gradients of the constraint functions. G is related to the width of the convex program: for linear constraints, the gradients are simply the coefficients of the constraints, and the width is the largest coefficient. Hence, G is at most $\sqrt{n}$ times the width. In section (3) we prove the following Theorem.

Theorem 3 (Main 1). *There exists an algorithm that for any $\varepsilon > 0$, returns a ε-approximate solution to mathematical program (1). The algorithm makes at most $\tilde{O}(\frac{G^2}{H} \cdot \frac{1}{\varepsilon})$ calls to* SEPARATION ORACLE, *and requires a single gradient computation and additional $O(n)$ time between successive oracle calls.*

Remark: Commonly the gradient of a given function can be computed in time which is linear in the function representation. Examples of functions which admit linear-time gradient computation include polynomials, logarithmic functions and exponentials.

The separation oracle which our algorithm invokes is defined as

$$\forall x \in \mathbb{S}_n . \text{SEPARATION ORACLE } (x) \triangleq \begin{cases} j \in [m] \text{ s.t } f_j(x) > \varepsilon & \text{if exists such } f_j \\ \\ FAIL & \text{otherwise} \end{cases}$$

If the constraints are given explicitly, often this oracle is easy to implement in time linear in the input size. Such constraints include linear functions, polynomials and logarithms. This oracle is also easy to implement in parallel: the constraints can be distributed amongst the available processors and evaluated in parallel.

For all cases in which H is zero or too small the theorem above cannot be applied. However, we can apply a simple reduction to strictly convex constraints and obtain the following corollary.

Corollary 4. *For any $\varepsilon > 0$, there exists an algorithm that returns a ε-approximate solution to mathematical program (1). The algorithm makes at most $\tilde{O}(\frac{G^2}{\varepsilon^2})$ calls to* SEPARATION ORACLE *and requires additional $O(n)$ time and a single gradient computation between successive oracle calls.*

---

[1] we denote $A \succeq B$ if the matrix $A - B \succeq 0$ is positive semi-definite

In comparison to Theorem 1, this corollary may require $O(n)$ more iterations. However, each iteration requires a call to SEPARATION ORACLE, as opposed to OPTIMIZATION ORACLE. A SEPARATION ORACLE requires only function evaluation, which can many times be implemented in linear time in the input size, whereas an OPTIMIZATION ORACLE could require expensive operations such as matrix inversions.

There is yet another alternative to deal with linear constraints and yet obtain linear dependence on ε. This is given by the following theorem. The approximation algorithm runs in time linear in $\frac{1}{\varepsilon}$, and yet does not require a lower bound on H. The downside of this algorithm is the computation of "generalized projections". A generalized projection of a vector $y \in \mathbb{R}^n$ onto a convex set $\mathcal{P}$ with respect to PSD matrix $A \succeq 0$ is defined to be $\prod_{\mathcal{P}}^A(y) = \arg\min_{x \in \mathcal{P}} (x - y)^\top A(x - y)$. Generalized projections can be cast as convex mathematical programs. If the underlying set is simple, such as the ball or simplex, then the program reduces to a convex quadratic program.

Theorem 5 (Main 2). *There exists an algorithm that for any $\varepsilon > 0$ returns a ε-approximate solution to mathematical program* (1). *The algorithm makes at most $\tilde{O}(\frac{nG}{\varepsilon})$ calls to* SEPARATION ORACLE *and requires computation of a generalized projection onto $\mathbb{S}_n$, a single gradient computation and additional $\tilde{O}(n^2)$ time between successive oracle calls.*

An example of an application of the above theorem is the following linear program.

$$\forall j \in [m] \ . \ A_j \cdot x \geq 0, \quad x \in \mathbb{S}_n \tag{2}$$

It is shown in [DV04] that general linear programming can be reduced to this form, and that without loss of generality, $\forall j \in [m] \ \|A_j\| = 1$. This format is called the "perceptron" format for linear programs. As a corollary to Theorem 5, we obtain

Corollary 6. *There exists an algorithm that for any $\varepsilon > 0$ returns a ε-approximate solution to linear program* (2). *The algorithm makes $\tilde{O}(\frac{n}{\varepsilon})$ iterations. Each iteration requires $\tilde{O}(n(m + n))$ computing time plus computation of a generalized projection onto the simplex.*

Theorem 5 and Corollary 6 extend the result of Bienstock and Iyengar [BI04] to general convex programming [2]. The running time of the algorithm is very similar to theirs: the number of iterations is the same, and each iteration also requires to solve convex quadratic programs (generalized projections onto the simplex in our case). Our algorithm is very different from [BI04]. The analysis is simpler, and relies on recent results from online learning. We note that the algorithm of Bienstock and Iyengar allows improved running time for sparse instances, whereas our algorithm currently does not.

1.2 Lagrangian relaxation and solving zero sum games

The relation between lagrangian relaxation and solving zero sum games was implicit in the original PST work, and explicit in the work of Freund and Schapire on online game playing [FS99] (the general connection between zero sum games and linear programming goes back to von Neumann).

Most previous lagrangian relaxation algorithms can be viewed as reducing the optimization problem at hand to a zero sum game, and then applying a certain online game playing algorithm, the Multiplicative Weights algorithm, to solve the game.

Our main insight is that the Multiplicative Weights algorithm can be replaced by any *online convex optimization* (see next section for precise definition) algorithm. Recent developments in

---

[2]Bienstock and Iyengar's techniques can also be extended to full linear programming by introducing dependence on the width which is similar to that of our algorithms [Bie06].

online game playing introduce algorithms with much better performance guarantees for online games with convex payoff functions [AH05, HKKA06]. Our results are derived by reducing convex optimization problems to games with payoffs which stem from convex functions, and using the new algorithms to solve these games.

The online framework also provides an alternative explanation to the aforementioned Klein and Young $\Omega(\varepsilon^{-2})$ lower bound on the number of iterations required by Frank-Wolfe algorithms to produce an ε-approximate solution. Translated to the online framework, previous algorithm were based on online algorithms with $\Omega(\sqrt{T})$ *regret* (the standard performance measure for online algorithms, see next section for precise definition). Our linear dependance on $\frac{1}{\varepsilon}$ is the consequence of using of online algorithms with $O(\log T)$ regret. This is formalized in Appendix A.

2 The general scheme

We outline a general scheme for approximately solving convex programs using online convex optimization algorithms. This is a generalization of previous methods which also allows us to derive our results stated in the previous section.

For this section we consider the following general mathematical program, which generalizes (1) by allowing an arbitrary convex set $\mathcal{P}$.

$$f_j(x) \leq 0 \quad \forall j \in [m] \tag{3}$$
$$x \in \mathcal{P}$$

In order to approximately solve (3), we reduce the mathematical problem to a game between two players: a *primal player* who tries to find a feasible point and the *dual player* who tries to disprove feasibility. This reduction is formalized in the following definition.

Definition 1. *The associated game with mathematical program (3) is between a primal player that plays $x \in \mathcal{P}$ and a dual player which plays a distribution over the constraints $p \in S_m$. For a point played by the primal player and a distribution of the dual player, the loss that the primal player incurs (and the payoff gained by the dual player) is given by the following function*

$$\forall \ x \in \mathcal{P} \ , p \in \mathbb{S}_m \ . \ \ g(x,p) \triangleq \sum_j p_j f_j(x)$$

The value of this game is defined to be $\lambda^* \triangleq \min_{x \in \mathcal{P}} \max_{p \in \mathbb{S}_m} g(x,p)$. Mathematical program (3) is feasible iff $\lambda^* \leq 0$.

By the above reduction, in order to check feasibility of mathematical program (3), it suffices to compute the value of the associated game λ^*. Notice that the game loss/payoff function g is smooth over the convex sets S_m and $\mathcal{P}$, linear with respect to p and convex with respect to x. For such functions, generalizations to the von Neumann minimax theorem, such as [Sio58] [3] imply that

$$\lambda^* = \min_{x \in \mathcal{P}} \max_{p \in \mathbb{S}_m} g(x,p) = \max_{p \in \mathbb{S}_m} \min_{x \in \mathcal{P}} g(x,p)$$

This suggests a natural approach to evaluate λ^*: simulate a repeated game between the primal and dual players such that in each iteration the game loss/payoff is determined according to the function g. In the simulation, the players play according to an online algorithm.

---

[3]All algorithms and theorems in this paper can be proved without relying on this minimax theorem. In fact, our results provide a new algorithmic proof of the generalized min-max theorem which is included in Appendix B.

The online algorithms we consider fall into the *online convex optimization* framework [Zin03], in which there is a fixed convex compact feasible set $\mathcal{P} \subset \mathbb{R}^n$ and an *arbitrary, unknown* sequence of convex cost functions $f_1, f_2, \dots : \mathcal{P} \to \mathbb{R}$. The decision maker must make a sequence of decisions, where the tth decision is a selection of a point $x_t \in \mathcal{P}$ and there is a cost of $f_t(x_t)$ on period t. However, x_t is chosen with only the knowledge of the set $\mathcal{P}$, previous points $x_1, \dots, x_{t-1}$, and the previous functions $f_1, \dots, f_{t-1}$. The standard performance measure for online convex optimization algorithms is called *regret* which is defined as:

$$\text{Regret}(\mathcal{A}, T) \triangleq \sup_{f_1, \dots, f_T} \left\{ \sum_{t=1}^{T} f_t(x_t) - \min_{x^* \in \mathcal{P}} \sum_{t=1}^{T} f_t(x^*) \right\} \tag{4}$$

We say that an algorithm $\mathcal{A}$ has *low regret* if $\text{Regret}(\mathcal{A}, T) = o(T)$. Later, we use to the procedure ONLINEALG, by which we refer to any low regret algorithm for this setting.

Another crucial property of online convex optimization algorithms is their running time. The running time is the time it takes to produce the point $x_t \in \mathcal{P}$ given all prior game history.

The running time of our approximate optimization algorithms will depend on these two parameters of online game playing algorithms: regret and running time. In Appendix C we survey some of the known online convex optimization algorithms and their properties.

We suggest three methods for approximating (3) using the approach outlined above. The first "meta algorithm" (it allows freedom in choice for the implementation of the online algorithm) is called PRIMALGAMEOPT and depicted in figure 1. For this approach, the dual player is simulated by an optimal adversary: at iteration t it plays a dual strategy p_t that achieves at least the game value λ^* (this reduces exactly to SEPARATION ORACLE).

The implementation of the primal player is an online convex optimization algorithm with low regret, which we denote by ONLINEALG. This online convex optimization algorithm produces decisions which are points in the convex set $\mathcal{P}$. The cost functions $f_1, f_2, \dots : \mathcal{P} \to \mathbb{R}$ are determined by the dual player's distributions. At iteration t, if the distribution output by the dual player us p_t, then the cost function to the online player is

$$\forall x \in \mathcal{P} \ . \ f_t(x) \triangleq g(x, p_t)$$

The low-regret property of the online algorithm used ensures that in the long run, the average strategy of the primal player will converge to the optimal strategy. Hence the average loss will converge to λ^*.

The "dual" version of this approach, in which the dual player is simulated by an online algorithm and the primal by an oracle, is called DUALGAMEOPT. In this case, the adversarial implementation of the primal player reduces to OPTIMIZATION ORACLE. The dual player now plays according to an online algorithm ONLINEALG. This online algorithm produces points in the m-dimensional simplex - the set of all distributions over the constraints. The payoff functions are determined according to the decisions of the primal player: at iteration t, if primal player produced point $x_t \in \mathcal{P}$, the payoff function is

$$\forall p \in \mathbb{S}_m \ . \ f_t(p) \triangleq g(x_t, p)$$

We also explore a third option, in which both players are implemented by online algorithms. This is called the PRIMALDUALGAMEOPT meta-algorithm. Pseudo-code for all versions is given in figure (1).

The following theorem shows that all these approaches yield an ε-approximate solution when the online convex optimization algorithm used to implement ONLINEALG has low regret.

PRIMALGAMEOPT (ε)
Let $t \leftarrow 1$. While Regret(ONLINEALG,t) $\geq \varepsilon t$ do

- Let $x_t \leftarrow$ ONLINEALG $(p_1, ..., p_{t-1})$.

- Let $j \leftarrow$ SEPARATION ORACLE (x_t). If $FAIL$ return x_t. Let $p_t \leftarrow e_j$, where e_j is the j'th standard basis vector of $\mathbb{R}^n$.

- $t \leftarrow t + 1$

Return $\bar{p} = \frac{1}{T} \sum_{t=1}^{T} p_t$

DUALGAMEOPT (ε)
Let $t \leftarrow 1$. While Regret(ONLINEALG,t) $\geq \varepsilon t$ do

- Let $p_t \leftarrow$ ONLINEALG $(x_1, ..., x_{t-1})$.

- Let $x_t \leftarrow$ OPTIMIZATION ORACLE (p_t). If $FAIL$ return p_t.

- $t \leftarrow t + 1$

Return $\bar{x} \triangleq \frac{1}{T} \sum_{t=1}^{T} x_t$

PRIMALDUALGAMEOPT (ε)
Let $t \leftarrow 1$. While Regret(ONLINEALG,t) $\geq \frac{\varepsilon}{2} t$ do

- Let $x_t \leftarrow$ ONLINEALG $(p_1, ..., p_{t-1})$.

- Let $p_t \leftarrow$ ONLINEALG $(x_1, ..., x_{t-1})$.

- $t \leftarrow t + 1$

If $\bar{x} \triangleq \frac{1}{T} \sum_{t=1}^{T} x_t$ is ε-approximate return $\bar{x}$. Else, return $\bar{p} = \frac{1}{T} \sum_{t=1}^{T} p_t$.

Figure 1: meta algorithms for approximate optimization by online game playing

Theorem 7. *Suppose OnlineAlg is an online convex optimization algorithm with low regret. If a solution to mathematical program (3) exists, then meta-algorithms* PRIMALGAMEOPT, DUAL-GAMEOPT *and* PRIMALDUALGAMEOPT *return an ε-approximate solution. Otherwise,* PRIMAL-GAMEOPT *and* DUALGAMEOPT *return a dual solution proving that the mathematical program is infeasible, and* PRIMALDUALGAMEOPT *returns a dual solution proving the mathematical program to be ε-close to being infeasible.*

Further, a ε-approximate solution is returned in $O(\frac{R}{\varepsilon})$ iterations, where $R = R(OnlineAlg, \varepsilon)$ is the smallest number T which satisfies the inequality $Regret(OnlineAlg, T) \leq \varepsilon T$.

Proof. **Part 1: correctness of** PRIMALGAMEOPT

If at iteration t SEPARATION ORACLE returns $FAIL$, then by definition of SEPARATION ORACLE,

$$\forall p^* \; . \; g(x_t, p^*) \leq \varepsilon \; \Rightarrow \; \forall j \in [m] \; . \; f_j(x_t) \leq \varepsilon$$

implying that x_t is a ε-approximate solution.

Otherwise, for every iteration $g(x_t, p_t) > \varepsilon$, and we can construct a dual solution as follows. Since the online algorithm guarantees sub-linear regret, for some iteration T the regret will be $R \leq \varepsilon T$. By definition of regret we have for any strategy $x^* \in \mathcal{P}$,

$$\varepsilon < \frac{1}{T} \sum_{t=1}^{T} g(x_t, p_t) \leq \frac{1}{T} \sum_{t=1}^{T} g(x^*, p_t) + \frac{R}{T} \leq \frac{1}{T} \sum_{t=1}^{T} g(x^*, p_t) + \varepsilon \leq g(x^*, \bar{p}) + \varepsilon$$

Where the last inequality is by the concavity (linearity) of $g(x, p)$ with respect to p Thus,

$$\forall x^* \, . \, g(x^*, \bar{p}) > 0$$

Hence $\bar{p}$ is a dual solution proving that the mathematical program is infeasible.

Part 2: correctness of DUALGAMEOPT The proof of this part is analogous to the first, and given in the full version of this paper.

If for some iteration t OPTIMIZATION ORACLE returns $FAIL$. According to the definition of OPTIMIZATION ORACLE,

$$\forall x \in \mathcal{P} \, . \, g(x, p_t) > 0$$

implying that p_t is a dual solution proving the mathematical program to be infeasible.

Else, in every iteration $g(x_t, p_t) \leq 0$. As before, for some iteration T the regret of the online algorithm will be $R \leq \varepsilon T$. By definition of regret we have (note that this time the online player wants to maximize his payoff)

$$\forall p^* \in P(\mathcal{F}) \, . \, 0 \geq \frac{1}{T} \sum_{t=1}^{T} g(x_t, p_t) \geq \frac{1}{T} \sum_{t=1}^{T} g(x_t, p^*) - \frac{R}{T} \geq \frac{1}{T} \sum_{t=1}^{T} g(x_t, p^*) - \varepsilon$$

Changing sides and using the convexity of the function $g(x, p)$ with respect to x (which follows from the convexity of the functions $f \in \mathcal{F}$) we obtain (for $\bar{x} = \frac{1}{T} \sum_{t=1}^{T} x_t$)

$$\forall p^* \in P(\mathcal{F}) \, . \, g(\bar{x}, p^*) \leq \frac{1}{T} \sum_{t=1}^{T} g(x_t, p^*) \leq \varepsilon$$

Which in turn implies that

$$\forall f \in \mathcal{F} \, . \, f(\bar{x}) \leq \varepsilon$$

Hence $\bar{x}$ is a ε-approximate solution.

Part 3: correctness of PRIMALDUALGAMEOPT

Denote R_1, R_2 the regrets attained by both online algorithms respectively. Using the low regret properties of the online algorithms we obtain for any x^*, p^*

$$\forall x^*, p^* \, . \, \sum_{t=1}^{T} g(x_t, p^*) - R_1 \leq \sum_{t=1}^{T} g(x_t, p_t) \leq \sum_{t=1}^{T} g(x^*, p_t) + R_2 \tag{5}$$

Let x^* be such that $\forall p \in P(\mathcal{F}) \, . \, g(x^*, p) \leq \lambda^*$. By convexity of $g(x, p)$ with respect to x,

$$\forall p^* \, . \, g(\bar{x}, p^*) \leq \frac{1}{T} \sum_{t=1}^{T} g(x_t, p^*) \leq \frac{1}{T} \sum_{t=1}^{T} g(x^*, p_t) + \frac{R_2 + R_1}{T} \leq \lambda^* + \varepsilon$$

Similarly, let p^* be such that $\forall x \in \mathcal{P} \ . \ g(x, p^*) \geq \lambda^*$. Then by concavity of g with respect to p and equation 5 we have

$$\forall x^* \ . \ g(x^*, \bar{p}) \geq \frac{1}{T} \sum_{t=1}^{T} g(x^*, p_t) \geq \frac{1}{T} \sum_{t=1}^{T} g(x_t, p^*) - \frac{R_2 + R_1}{T} \geq \lambda^* - \varepsilon$$

Hence, if $\lambda^* \leq 0$, then $\bar{x}$ satisfies

$$\forall p^* \ . \ g(\bar{x}, p^*) \leq \varepsilon \ \Rightarrow \ \forall j \in [m] \ . \ f_j(\bar{x}) \leq \varepsilon$$

And hence is a ε-approximate solution. Else,

$$\forall x^* \ . \ g(x^*, \bar{p}) > -\varepsilon$$

And $\bar{p}$ is a dual solution proving that the following mathematical program is infeasible.

$$f_j(x) \leq -\varepsilon \quad \forall j \in [m]$$
$$x \in \mathcal{P}$$

$\square$

3 Applications

3.1 Strictly convex programs

We start with the easiest and perhaps most surprising application of Theorem 7. Recall that the feasibility problem we are considering:

$$f_j(x) \leq 0 \quad \forall j \in [m] \tag{6}$$
$$x \in \mathbb{S}_n$$

Where the functions $\{f_j\}$ are all strictly convex such that $\forall x \in \mathbb{S}_n, j \in [m] \ . \ \nabla^2 f_j(x) \succeq H \cdot I_n$ and $\|\nabla f_j(x)\|_2 \leq G$

Proof of Theorem 3. Consider the associated game with value

$$\lambda^* \triangleq \min_{x \in \mathbb{S}_n} \max_{j \in [m]} f_j(x)$$

The convex problem is feasible iff $\lambda^* \leq 0$. To approximate λ^*, we apply the PRIMALGAMEOPT meta algorithm. In this case, the vectors x_t are points in the simplex, and p_t are distributions over the constraints. The online algorithm used to implement ONLINEALG is Online Convex Gradient Descent (OCGD). The resulting algorithm is strikingly simple, as depicted in figure 2.

According to Theorem 1 in [HKKA06], the regret of OCGD is bounded by $Regret(T) = O(\frac{G^2}{H} \log T)$. Hence, the number of iterations till the regret drops to εT is $\tilde{O}(\frac{G^2}{H} \frac{1}{\varepsilon})$. According to Theorem 7, this is the number of iterations required to obtain an ε-approximation.

In each iteration, the OCGD algorithm needs to update the current online strategy (the vector x_t) according to the gradient and project onto $\mathbb{S}_n$. This requires a single gradient computation. A projection of a vector $y \in \mathbb{R}^n$ onto $\mathbb{S}_n$ is defined to be $\prod_{\mathcal{P}}(y) = \arg\min_{x \in \mathbb{S}_n} \|x - y\|_2$. The projection of a vector onto the simplex can be computed in time $\tilde{O}(n)$ (see procedure SIMPLEXPROJECT described in Appendix D). Other than the gradient computation and projection, the running time of OCGD is $O(n)$ per iteration. $\square$

STRICTLYCOVEXOPT.

Input: Instance in format (6), parameters G, H approximation guarantee ε.

Let $t \leftarrow 1$, $x_1 \leftarrow \frac{1}{n}\vec{1}$.

While $t \leq \frac{G^2}{H}\frac{1}{\varepsilon}\log\frac{1}{\varepsilon}$ do

- Let $j \leftarrow$ SEPARATION ORACLE (x_t) (i.e. an index of a violated constraint). If all constraints are satisfied return x_t. Else, let $\nabla_{t-1} = \nabla f_j(x_{t-1})$. Let $p_t \leftarrow e_j$ where e_j is the j'th standard basis vector of $\mathbb{R}^n$.

- Set $y_t = x_{t-1} - \frac{1}{H \cdot t}\nabla_{t-1}$

- Set $x_t =$ SIMPLEXPROJECT (y_t).

- $t \leftarrow t+1$

Return $\bar{p} = \frac{1}{T}\sum_{t=1}^{T} p_t$

Figure 2: An approximation algorithm for strictly convex programs. Here $\vec{1}$ stands for the vector with one in all coordinates.

Remark: It is clear that the above algorithm can be applied the more general version of convex program (3), where the simplex is replaced by an arbitrary convex set $\mathcal{P} \subseteq \mathbb{R}^n$. The only change required is in the projection step. For Theorem 3, we assumed the underlying convex set is the simplex, hence the projection can be computed in time $\tilde{O}(n)$. Projections can be computed in linear time also for the hypercube and ball. For convex sets which are intersections of hyperplanes (or convex parabloids), computing a projection reduces to optimizing a convex quadratic function over linear (quadratic) constraints. These optimization problems allow for more efficient algorithms than general convex optimization [LVBL98].

As a concrete example of the application of Theorem 3, consider the case of strictly convex quadratic programming. In this case, there are m constraint functions of the form $f_j(x) = x^\top A_j x + b_j^\top x + c$, where the matrices A_j are positive-definite. If $A_j \succeq H \cdot I$, and $\forall_{x \in \mathbb{S}_n}\|A_j x + b_j\|_2 \leq G$, then Theorem 3 implies that an ε-approximate solution can be found in $\tilde{O}(\frac{G^2}{H\varepsilon})$ iterations.

The implementation of SEPARATION ORACLE involves finding a constraint violated by more than ε. In the worst case all constrains need be evaluated in time $O(mn^2)$. The gradient of any constraint can be computed in time $O(n^2)$. Overall, the time per SEPARATION ORACLE computation is $\tilde{O}(mn^2)$. We conclude that the total running time to obtain a ε-approximation solution is $\tilde{O}(\frac{G^2mn^2}{H\varepsilon})$. Notice that the input size is mn^2 in this case.

3.2 Linear and Convex Programs

In this section we prove Theorem 5, which gives an algorithm for convex programming that has running time proportional to $\frac{1}{\varepsilon}$. As a simple consequence we obtain corollary 6 for linear programs. The algorithm is derived using the PRIMALGAMEOPT meta-algorithm and the ONLINE NEWTON STEP (ONS) online convex optimization algorithm (see appendix C) to implement ONLINEALG. The resulting algorithm is described in figure 3 below.

Since for general convex programs the constraints are not strictly convex, one cannot apply online algorithms with logarithmic regret directly as in the previous subsection. Instead, we first perform a reduction to a mathematical program with exp-concave constraints, and then approxi-

CovexOpt.

Input: Instance in format (7), parameters G, D, ω approximation guarantee ε.

Let $t \leftarrow 1$, $x_1 \leftarrow \frac{1}{n}\vec{1}$, $\beta \leftarrow \frac{1}{2}\min\{1, \frac{1}{4GD}\}$, $A_0 \leftarrow \frac{1}{D^2\beta^2}I_n$, $A_0^{-1} \leftarrow D^2\beta^2 I_n$.

While $t \leq 6nGD\frac{1}{\varepsilon}\log\frac{1}{\varepsilon}$ do

- Let $j \leftarrow$ SEPARATION ORACLE (x_t) (i.e. an index of a violated constraint). If all constraints are satisfied return x_t. Else, let $\nabla_{t-1} = \nabla\{\log(e + \omega^{-1}f_j(x))\}$ and $p_t \leftarrow e_j$ where e_j is the j'th standard basis vector of $\mathbb{R}^n$.

- Set $y_t = x_{t-1} + \frac{1}{\beta}A_{t-1}^{-1}\nabla_{t-1}$

- Set $x_t = \arg\min_{x\in\mathcal{P}} (y_t - x)^\top A_{t-1}(y_t - x)$

- Set $A_t = A_{t-1} + \nabla_{t-1}\nabla_{t-1}^\top$, and $A_t^{-1} = A_{t-1}^{-1} - \frac{A_{t-1}^{-1}\nabla_{t-1}\nabla_{t-1}^\top A_{t-1}^{-1}}{1+\nabla_{t-1}^\top A_{t-1}^{-1}\nabla_{t-1}}$

Return $\bar{p} = \frac{1}{T}\sum_{t=1}^T p_t$

Figure 3: An approximation algorithm for convex programs. Here I_n stands for the n-dimensional identity matrix.

mate the reduced instance.

Proof of Theorem 5. In this proof it is easier for us to consider concave constraints rather than convex. Mathematical program (1) can be converted to the following by negating each constraint:

$$f_j(x) \geq 0 \quad \forall j \in [m] \tag{7}$$
$$x \in \mathcal{P}$$

where the functions $\{f_j\}$ are all concave such that $\forall x \in \mathcal{P}, j \in [m]$. $\|\nabla f_j(x)\|_2 \leq G$ and $\forall x \in \mathcal{P}, j \in [m]$. $|f_j(x)| \leq \omega$. This program is even more general than (1) as it allows for an arbitrary convex set $\mathcal{P}$ rather than $\mathbb{S}_n$.

Let $\rho = \max_{x\in\mathcal{P}}\min_j\{f_j(x)\}$. The question to whether this convex program is feasible is equivalent to whether $\rho > 0$.

In order to approximately solve this convex program, we consider a different concave mathematical program,

$$\log(e + \omega^{-1}f_j(x)) \geq 1 \quad \forall j \in [m] \tag{8}$$
$$x \in \mathcal{P}$$

It is a standard fact that concavity is preserved for the composition of a non-decreasing concave function with another concave function, i.e. the logarithm of positive concave functions is itself concave. To solve this program we consider the (non-linear) zero sum game defined by the following min-max formulation

$$\lambda^* \triangleq \max_{x\in\mathcal{P}}\min_{j\in[m]} \log(e + \omega^{-1}f_j(x)) \tag{9}$$

The following two claims show that program (8) is closely related to (7).

Claim 8. $\lambda^* = \log(e + \omega^{-1}\rho)$.

11

Proof. Let x be a solution to (7) which achieves the value ρ, that is $\forall j \in [m] \,.\, f_j(x) \geq \rho$. This implies that $\forall j \in [m] \,.\, \log(e + \omega^{-1} f_j(x)) \geq \log(e + \omega^{-1}\rho)$, and in particular $\forall q \quad g(x,q) \geq \log(e + \omega^{-1}\rho)$ hence $\lambda^* \geq \log(e + \omega^{-1}\rho)$.

For the other direction, suppose that $\lambda^* = \log(e + z) > \log(e + \omega^{-1}\rho)$ for some $z > \omega^{-1}\rho$. Then there exists an x such that $\forall j \in [m] \,.\, \log(e + \omega^{-1} f_j(x)) \geq \lambda^* > \log(e + z)$ or equivalently $\forall j \in [m] \,.\, f_j(x) \geq z > \rho$ in contradiction to the definition of ρ. $\qquad\square$

Claim 9. *An ε-approximate solution for* (8) *is a $3\omega\varepsilon$-approximate solution for* (7).

Proof. A ε-approximate solution to (8) satisfies $\forall j \,.\, \log(e + \omega^{-1} f_j(x)) \geq \lambda^* - \varepsilon = \log(e + \omega^{-1}\rho) - \varepsilon$. Therefore, by monotonicity of the logarithm we have

$$
\begin{aligned}
\omega^{-1} f_j(x) \quad &\geq e^{\log(e + \omega^{-1}\rho) - \varepsilon} - e \\
&= (e + \omega^{-1}\rho) \cdot e^{-\varepsilon} - e \\
&\geq (e + \omega^{-1}\rho)(1 - \varepsilon) - e \quad \text{since } e^{-x} \geq 1 - x \\
&= \omega^{-1}\rho(1 - \varepsilon) - e\varepsilon
\end{aligned}
$$

Which implies

$$
f_j(x) \geq \rho(1 - \varepsilon) - 3\omega\varepsilon
$$

$\square$

We proceed to approximate λ^* using PRIMALGAMEOPT and choose the ONLINE NEWTON STEP (ONS) algorithm (see appendix C) as ONLINEALG. The resulting algorithm is depicted in figure 3.

We note that here the primal player is maximizing payoff as opposed to the minimization version in the proof of Theorem 7. The maximization version of Theorem 7 can be proved analogously.

In order to analyze the number of iterations required, we calculate some parameters of the constraints of formulation (8). See appendix C for explanation on how the different parameters effect the regret and running time of ONLINE NEWTON STEP.

The constraint functions are 1-exp-concave, since their exponents are linear functions. Their gradients are bounded by

$$
\tilde{G} \triangleq \max_{j \in m} \max_{x \in \mathcal{P}} \|\nabla \log(e + \omega^{-1} f_j(x))\| = \max_{j \in m} \max_{x \in \mathcal{P}} \left\| \frac{\omega^{-1}\nabla f_j(x)}{e + \omega^{-1} f_j(x)} \right\| \leq \omega^{-1} G
$$

According to Theorem 2 in [HKKA06], the regret of ONS is $O((\frac{1}{\alpha} + GD)n \log T)$. In our setting, $\alpha = 1$ and G is replaced by $\tilde{G}$. Therefore, the regret becomes smaller then εT after $O(\frac{nGD\omega^{-1}}{\varepsilon})$ iterations. By Theorem 7, after $T = \tilde{O}(\frac{nGD\omega^{-1}}{\delta})$ iterations we obtain an δ-approximate solution, i.e a solution x^* such that

$$
\min_{j \in [m]} \log(e + \omega^{-1} f_j(x^*)) \geq \lambda^* - \delta
$$

Which by claim 9 is a $3\omega\delta$-approximate solution to the original math program. Taking $\delta = O(\omega^{-1}\varepsilon)$ we obtain an ε-approximate solution to concave program (7) in $T = \tilde{O}(\frac{nGD}{\varepsilon})$ iterations.

We now analyze the running time per iteration. Each iteration requires a call to SEPARATION ORACLE in order to find an ε-violated constraint. The gradient of the constraint need be computed. According to the gradient the ONS algorithm takes $O(n^2)$ time to update its internal data structures (which are y_t, A_t, A_t^{-1} in figure 3). Finally ONS computes a generalized projection onto $\mathcal{P}$, which corresponds to computing $\arg\min_{x \in \mathcal{P}} (y - x)^\top A_{t-1}(y - x)$ given y (see appendix D)

If $\mathcal{P} = \mathbb{S}_n$, then $D = 1$ and the bounds of Theorem 5 are met.

$\square$

Given Theorem 5, it is straightforward to derive corollary 6 for linear programs:

proof of Corollary 6. For linear programs in format (2), the gradients of the constraints are bounded by $\max_{j \in [m]} \|A_j\| \leq 1$. In addition, SEPARATION ORACLE is easy to implement in time $O(mn)$ by evaluating all constraints.

Denote by T_{proj}^S the time to compute a generalized projection onto the simplex. A worst case bound is $T_{proj}^S = O(n^3)$, using interior point methods (this is an instance quadratically constrained convex quadratic program, see [LVBL98]).

Plugging these parameters into Theorem 5, the total running time comes to

$$\tilde{O}(\frac{n}{\varepsilon} \cdot (nm + n^2 + T_{proj}^S))$$

□

Remark: As is the case for strictly convex programming, our framework actually provides a more general algorithm that requires a SEPARATION ORACLE. Given such an oracle, the corresponding optimization problem can be solved in time $\tilde{O}(\frac{n}{\varepsilon} \cdot (n^2 + T_{A,proj} + T_{oracle}))$ where T_{oracle} is the running time of SEPARATION ORACLE.

3.3 Derivation of previous results

For completeness, we prove Theorem 1 using our framework. Even more generally, we prove the theorem for general convex program (3) rather than (1).

Proof of Theorem 1. Consider the associated game with value

$$\lambda^* \triangleq \min_{x \in \mathcal{P}} \max_{j \in [m]} f_j(x) = \max_{p \in S_m} \min_{x \in \mathcal{P}} \sum_{i=1}^m p_i f_i(x)$$

The convex problem is feasible iff $\lambda^* \leq 0$. To approximate λ^*, we apply the DUALGAMEOPT meta algorithm. The vectors x_t are points in the convex set $\mathcal{P}$, and p_t are distributions over the constraints, i.e. points in the m dimensional simplex. The payoff functions for ONLINEALG in iteration t are of the form

$$\lambda p \; . \; g(x_t, p) = \sum_i p_i f_i(x_t)$$

The online algorithm used to implement ONLINEALG is the Multiplicative Weights algorithm (MW). According to Theorem 13 in appendix C, the regret of MW is bounded by $\text{Regret}_T(MW) = O(G_\infty \sqrt{T \log m})$ (the dimension of the online player is m in this case). Hence, the number of iterations till the regret drops to εT is $\tilde{O}(\frac{G_\infty^2}{\varepsilon^2})$. According to Theorem 7, this is the number of iterations required to obtain an ε-approximation.

To bound G_∞, note that the payoff functions $\lambda p \; . \; g(x_t, p)$ are linear. Their gradients are m-dimensional vectors such that the i'th coordinate is the value of the i'th constraint on the point x_t, i.e. $f_i(x_t)$. Thus, the ℓ_∞ norm of the gradients can be bounded by

$$G_\infty = \max_{x \in \mathcal{P}} \max_{t \in [T]} \nabla(\lambda p \; . \; g(x_t, p)) \leq \max_{i \in [m]} \max_{x \in \mathcal{P}} f_i(x)$$

And the latter expression is bounded by the width $\omega = \max_{i \in [m]} \max_{x \in \mathcal{P}} |f_i(x)|$. Thus the number of iterations to obtain an ε-approximate solution is bounded by $\tilde{O}(\frac{\omega^2}{\varepsilon^2})$.

In each iteration, the MW algorithm needs to update the current online strategy (the vector p_t) according to the gradient in time $O(m)$. This requires a single gradient computation. □

4 Acknowledgements

References

[AH05]   Amit Agarwal and Elad Hazan. Efficient algorithms for online game playing and universal portfolio management. *ECCC TR06-033*, 2005.

[AHK05a] S. Arora, E. Hazan, and S. Kale. The multiplicative weights update method: a meta algorithm and applications. *Manuscript*, 2005.

[AHK05b] Sanjeev Arora, Elad Hazan, and Satyen Kale. Fast algorithms for approximate semide.nite programming using the multiplicative weights update method. In *46th IEEE FOCS*, pages 339–348, 2005.

[Ber06]   Dimitris Bertsimas. personal communications, 2006.

[BI04]   D. Bienstock and G. Iyengar. Solving fractional packing problems in oast(1/ε) iterations. In *STOC '04: Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 146–155, New York, NY, USA, 2004. ACM Press.

[Bie01]   D. Bienstock. Potential function methods for approximately solving linear programs: Theory and practice, 2001.

[Bie06]   Daniel Bienstock. personal communications, 2006.

[Cov91]   T. Cover. Universal portfolios. *Math. Finance*, 1:1–19, 1991.

[DS06]   M. Dudik and R. E. Schapire. Maximum entropy distribution estimation with generalized regularization. In *Proceedings of the 19th Annual Conference on Learning Theory*, pages 123–138, 2006.

[DV04]   John Dunagan and Santosh Vempala. A simple polynomial-time rescaling algorithm for solving linear programs. In *STOC '04: Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 315–320, New York, NY, USA, 2004. ACM Press.

[Fle00]   Lisa K. Fleischer. Approximating fractional multicommodity flow independent of the number of commodities. *SIAM J. Discret. Math.*, 13(4):505–520, 2000.

[FS99]   Y. Freund and R. E. Schapire. Adaptive game playing using multiplicative weights. *Games and Economic Behavior*, 29:79–103, 1999.

[FW56]   M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3:149–154, 1956.

[GK94]   Michael D. Grigoriadis and Leonid G. Khachiyan. Fast approximation schemes for convex programs with many block and coupling constraints. *SIAM Journal on Optimization*, 4:86–107, 1994.

[GK95]   M. Grigoriadis and L. Khachiyan. A sublinear-time randomized approximation algorithm for matrix games. In *Operations Research Letters*, volume 18, pages 53–58, 1995.

[GK98]   N. Garg and J. Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science(FOCS-98)*, pages 300–309, Los Alamitos, CA, November8–11 1998. IEEE Computer Society.

[HH06]   Eran Halperin and Elad Hazan. Haplofreq - estimating haplotype frequencies efficiently. *Journal of Computational Biology*, 13(2):481–500, 2006.

[HKKA06] Elad Hazan, Adam Kalai, Satyen Kale, and Amit Agarwal. Logarithmic regret algorithms for online convex optimization. *to appear in 19'th COLT*, 2006.

[HSSW96] David P. Helmbold, Robert E. Schapire, Yoram Singer, and Manfred K. Warmuth. On-line portfolio selection using multiplicative updates. In *ICML*, pages 243–251, 1996.

[Jan06]   Klaus Jansen. Approximation algorithm for the mixed fractional packing and covering problem. *SIAM J. on Optimization*, 17(2):331–352, 2006.

[Kha04]   Rohit Khandekar. *Lagrangian Relaxation based Algorithms for Convex Programming Problems*. PhD thesis, Indian Institute of Technology, Delhi, 2004. Available at `http://www.cse.iitd.ernet.in/~rohitk`.

[KL96]   Philip Klein and Hsueh-I. Lu. Efficient approximation algorithms for semidefinite programs arising from MAX CUT and COLORING. In *Proceedings of the twenty-eighth annual ACM Symposium on the Theory of Computing*, pages 338–347, 1996.

[KPST94]   Philip Klein, Serge Plotkin, Clifford Stein, and Eva Tardos. Faster approximation algorithms for the unit capacity concurrent flow problem with applications to routing and finding sparse cuts. *SIAM J. Comput.*, 23(3):466–487, 1994.

[KW97]   Jyrki Kivinen and Manfred K. Warmuth. Exponentiated gradient versus gradient descent for linear predictors. *Inf. Comput.*, 132(1):1–63, 1997.

[KY99]   Philip Klein and Neal Young. On the number of iterations for Dantzig-Wolfe optimization and packing-covering approximation algorithms. *Lecture Notes in Computer Science*, 1610:320–327, 1999.

[LSM+91]   Tom Leighton, Clifford Stein, Fillia Makedon, Éva Tardos, Serge Plotkin, and Spyros Tragoudas. Fast approximation algorithms for multicommodity flow problems. In *STOC '91: Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 101–111, New York, NY, USA, 1991. ACM Press.

[LVBL98]   Miguel Sousa Lobo, Lieven Vandenberghe, Stephen Boyd, and Herve Lebret. Applications of second-order cone programming, 1998.

[Nes04]   Y. Nesterov. Rounding of convex sets and efficient gradient methods for linear programming problems. Technical Report 4, CORE discussion paper, 2004.

[PST91]   Serge A. Plotkin, David B. Shmoys, and Tardos Tardos. Fast approximation algorithm for fractional packing and covering problems. In *Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science, FOCS'91 (San Juan, Puerto Rico,*

October 1-4, 1991), pages 495–504, Los Alamitos-Washington-Brussels-Tokyo, 1991. IEEE Computer Society Press.

[Sio58]   Maurice Sion. On general minimax theorems. *Pacific J. Math.*, 8:171–176, 1958.

[SM90]   Farhad Shahrokhi and David W. Matula. The maximum concurrent flow problem. *J. ACM*, 37(2):318–334, 1990.

[You95]   Neal E. Young. Randomized rounding without solving the linear program. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 170–178, San Francisco, California, 22–24 January 1995.

[Zin03]   Martin Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the Twentieth International Conference (ICML)*, pages 928–936, 2003.

A   Lower bounds

The algorithmic scheme described hereby generalizes previous approaches, which are generally known as **Dantzig-Wolfe-type** algorithms. These algorithms are characterized by the way the constraints of mathematical program (1) are accessed: every iteration only a single OPTIMIZATION ORACLE call is allowed.

For the special case in which the constraints are linear, there is a long line of work leading to tight lower bounds on the number of iterations required for algorithms within the Dantzig-Wolfe framework to provide an ε-approximate solution. Already in 1977, Khachiyan proved an $\Omega(\frac{1}{\varepsilon})$ lower bound on the number of iterations to achieve an error of ε. This was tightened to $\Omega(\frac{1}{\varepsilon^2})$ by Klein and Young [KY99], and independently by Freund and Schapire [FS99]. Some parameters were tightened in [AHK05a].

For the game theoretic framework we consider, it is particularly simple and intuitive to derive tight lower bounds. These lower bounds do **not** hold for the more general Dantzig-Wolfe framework. However, virtually all lagrangian-relaxation-type algorithms known can be derived from our framework. Thus, for all these algorithms lower bounds on the running time in terms of ε can be derived from the following observation.

In our setting, the number of iterations depends on the regret achievable by the online game playing algorithm which is deployed. Tight lower bounds are known on regret achievable by online algorithms.

Lemma 10 (folklore). *For linear payoff functions any online convex optimization algorithm incurs* $\Omega(G_\infty \sqrt{T})$ *regret.*

Proof. This can be seen by a simple randomized example. Consider $\mathcal{P} = [-1, 1]$ and linear functions $f_t(x) = r_t x$, where $r_t = \pm 1$ are chosen in advance, independently with equal probability. $\mathbf{E}_{r_t}[f_t(x_t)] = 0$ for any t and x_t chosen online, by independence of x_t and r_t. However, $\mathbf{E}_{r_1,\ldots,r_T}[\min_{x \in K} \sum_1^T f_t(x)] = \mathbf{E}[-|\sum_1^T r_t|] = -\Omega(\sqrt{T})$. Multiplying r_t by any constant (which corresponds to G_∞) yields the result. $\square$

The above simple lemma is essentially the reason why it took more than a decade to break the $\frac{1}{\varepsilon^2}$ running time. The reason why we obtain algorithms with linear dependance on ε is the use of strictly convex constraints (or, in case the original constraints are linear, apply a reduction to strictly convex constraints).

B A general min-max theorem

In this section prove a generalized version of the von Neumann min-max theorem. The proof is algorithmic in nature, and differs from previous approaches which were based on fixed point theorems.

Freund and Schapire [FS99] provide an algorithmic proof of the (standard) min-max theorem, and this proof is an extension of their ideas to the more general case. The additional generality is in two parameters: first, we allow more general underlying convex sets, whereas the standard min-max theorem deals with the n-dimensional simplex $\mathbb{S}_n$. Second, we allow convex-concave functions as defined below rather than linear functions. Both generalities stems from the fact that we use general online convex optimization algorithms as the strategy for the two players, rather than specific "expert-type" algorithms which Freund and Schapire use. Other than this difference, the proof itself follows [FS99] almost exactly.

The original minimax theorem can be stated as follows.

Theorem 11 (von Neumann). *If X, Y are finite dimensional simplices and f is a bilinear function on $X \times Y$, then f has a saddle point, i.e.*

$$\min_{x \in X} \max_{y \in Y} f(x, y) = \max_{y \in Y} \min_{x \in X} f(x, y)$$

Here we consider a more general setting, in which the two sets X, Y can be arbitrary closed, non-empty, bounded and convex sets in Euclidian space and the function f is convex-concave as defined by:

Definition 2. *A function f on $X \times Y$ is convex-concave if for every $y \in Y$ the function $\forall x \in X \ \ f_y(x) \triangleq f(x, y)$ is convex on X and for every $x \in X$ the function $\forall y \in Y \ \ f_x(y) \triangleq f(x, y)$ is concave on Y.*

Theorem 12. *If X, Y are closed non-empty bounded convex sets and f is a convex-concave function on $X \times Y$, then f has a saddle point, i.e.*

$$\max_{y \in Y} \min_{x \in X} f(x, y) = \min_{x \in X} \max_{y \in Y} f(x, y)$$

Proof. Let $\mu^* \triangleq \max_{y \in Y} \min_{x \in X} f(x, y)$ and $\lambda^* \triangleq \min_{x \in X} \max_{y \in Y} f(x, y)$. Obviously $\mu^* \leq \lambda^*$ (this is called *weak duality*).

Apply the algorithm PRIMALDUALGAMEOPT with any low-regret online convex optimization algorithm. [4] Then by the regret guarantees we have for the first algorithm (let $\bar{y} = \frac{1}{T} \sum_{t=1}^{T} y_t$)

$$
\begin{aligned}
\frac{1}{T} \sum_{t=1}^{T} f(x_t, y_t) \leq \ & \min_{x \in X} \tfrac{1}{T} \sum_{t=1}^{T} f(x, y_t) + \tfrac{R_1}{T} \\
\leq \ & \min_{x \in X} f(x, \bar{y}) + \tfrac{R_1}{T} && \text{concavity of } f_x \\
\leq \ & \max_{y \in Y} \min_{x \in X} f(x, y) + \tfrac{R_1}{T} \\
= \ & \mu^* + \tfrac{R_1}{T}
\end{aligned}
$$

---

[4]for a low-regret algorithm to exist, we need f to be convex-concave and the underlying sets X, Y to be convex, nonempty, closed and bounded.

Similarly for the second online algorithm we have (let $\bar{x} = \frac{1}{T} \sum_{t=1}^{T} x_t$)

$$
\begin{aligned}
\frac{1}{T} \sum_{t=1}^{T} f(x_t, y_t) \geq \quad & \max_{y \in Y} \tfrac{1}{T} \sum_{t=1}^{T} f(x_t, y) - \tfrac{R_1}{T} \\
\geq \quad & \min_{x \in X} f(\bar{x}, y) + \tfrac{R_1}{T} \qquad \text{convexity of } f_y \\
\geq \quad & \min_{x \in X} \max_{y \in Y} f(x, y) + \tfrac{R_1}{T} \\
= \quad & \lambda^* + \tfrac{R_1}{T}
\end{aligned}
$$

Combining both observations we obtain

$$
\lambda^* - \frac{R_2}{T} \leq \mu^* + \frac{R_1}{T}
$$

As $T \mapsto \infty$ we obtain $\mu^* \geq \lambda^*$.

$\square$

C   Online convex optimization algorithms

Figure (4) summarizes several known low regret algorithms. The running time is the time it takes to produce the point $x_t \in \mathcal{P}$ given all prior game history.

Algorithm	Regret bound	running time
Online convex gradient descent	$\frac{G_2^2}{H} \log(T)$	$O(n + T_{proj})$
Online Newton step	$(\frac{1}{\alpha} + G_2 D) n \log T$	$O(n^2 + T_{A,proj})$
Exponentially weighted online opt.	$\frac{1}{\alpha} n \log(T)$	$poly(n)$
Multiplicative Weights	$G_\infty \sqrt{T \log n}$	$O(n)$

Figure 4: Various online convex optimization algorithms and their performance. T_{proj} is the time to project a vector $y \in \mathbb{R}^n$ to $\mathcal{P}$, i.e. to compute $\arg\min_{x \in \mathcal{P}} \|y - x\|_2$. $T_{A,proj}$ is the time to project a vector $y \in \mathbb{R}^n$ to $\mathcal{P}$ using the norm defined by PSD matrix A, i.e. to compute $\arg\min_{x \in \mathcal{P}} (y - x)^\top A (y - x)$.

The first three algorithms are from [HKKA06] and are applicable to the general online convex optimization framework. The description and analysis of these algorithms is beyond our scope, and the reader is referred to the paper.

The last algorithm is based on the ubiquitous Multiplicative Weights Update method (for more applications of the method see survey [AHK05a]), and is provided below. Although it was used many times for various applications (for very detailed analysis in similar settings see [KW97]), this application to general online convex optimization over the simplex seems to be new (Freund and Schapire [FS99] analyze this algorithm exactly, although for linear payoff functions rather than for general convex functions).

This online algorithm, which is called "exponentiated gradient" in the machine learning literature, attains similar performance guarantees to the "online gradient descent" algorithm of Zinkevich [Zin03]. Despite being less general than Zinkevich's algorithm (we only give an application to the n-dimensional simplex, whereas online gradient descent can be applied over any convex set in Euclidian space), it attains somewhat better performance as given in the following theorem.

Multiplicative Weights.

Inputs: parameter $\eta < \frac{1}{2}$.

- On period 1, play the uniform distribution $x_1 = \vec{1} \in S_n$. Let $\forall i \in [n]$. $w_i^1 = 1$

- On period t, update

$$w_i^t = w_i^{t-1} \cdot (1 + \frac{\eta}{G_\infty} \nabla_{t-1}(i))$$

where $\nabla_t \triangleq \nabla f_t(x_t)$, and play x_t defined as

$$x_t \triangleq \frac{w^t}{\|w^t\|_1}$$

Figure 5: The Multiplicative Weights algorithm for online convex optimization over the simplex

Theorem 13. *The Multiplicative Weights algorithm achieves the following guarantee, for all $T \geq 1$.*

$$Regret(MW, T) = \sum_{t=1}^{T} f_t(x_t) - \min_{x \in S_n} \sum_{t=1}^{T} f_t(x) \leq O(G_\infty \sqrt{\log n} \sqrt{T})$$

Proof. Define $\Phi^t = \sum_i w_i^t$. Since $\frac{1}{G_\infty} \nabla_t(i) \in [0, 1]$,

$$
\begin{aligned}
\Phi^{t+1} &= \sum_i w_i^{t+1} = \sum_i w_i^t (1 - \frac{\eta}{G_\infty} \nabla_t(i)) \\
&= \Phi^t - \frac{\eta \Phi^t}{G_\infty} \sum_i x_t(i) \nabla_t(i) && \text{since } x_t(i) = w_i^t / \Phi^t \\
&= \Phi^t (1 - \eta x_t \nabla_t / G_\infty) \\
&\leq \Phi^t e^{-\eta x_t \nabla_t / G_\infty} && \text{since } 1 - x \leq e^{-x} \text{ for } |x| \leq 1
\end{aligned}
$$

After T rounds, we have

$$\Phi^T \leq \Phi^1 e^{-\eta \sum_t x_t \nabla_t / G_\infty} = n e^{-\eta \sum_t x_t \nabla_t / G_\infty} \tag{10}$$

Also, for every $i \in [n]$, using the following facts which follow immediately from the convexity of the exponential function

$$
\begin{aligned}
(1 - \eta)^x &\leq (1 - \eta x) && \text{if } x \in [0, 1] \\
(1 + \eta)^{-x} &\leq (1 - \eta x) && \text{if } x \in [-1, 0]
\end{aligned}
$$

We have

$$
\begin{aligned}
\Phi^T = \sum_t w_i^T &\geq w_i^T \\
&= \prod_t (1 - \eta \nabla_t(i) / G_\infty) \\
&\geq (1 - \eta)^{\sum_{t>0} \nabla_t(i)/G_\infty} (1 + \eta)^{\sum_{t<0} -\nabla_t(i)/G_\infty}
\end{aligned}
$$

where the subscripts ≥ 0 and < 0 refer to the rounds t where $\nabla_t(i)$ is ≥ 0 and < 0 respectively. So together with (10)

19

$$ne^{-\eta \sum_t x_t \nabla_t / G_\infty} \geq (1-\eta)^{\sum_{t>0} \nabla_t(i)/G_\infty}(1+\eta)^{\sum_{t<0} -\nabla_t(i)/G_\infty}$$

Taking logarithms and using $\ln(\frac{1}{1-\eta}) \leq \eta + \eta^2$ and $\ln(1+\eta) \geq \eta - \eta^2$ for $\eta \leq \frac{1}{2}$ we get for all $i \in [n]$ and $x^* \in S_n$

$$\sum_t x_t \nabla_t \leq (1+\eta)\sum_{\geq 0} \nabla_t(i) + (1-\eta)\sum_{<0} \nabla_t(i) + \frac{G_\infty \log n}{\eta} \leq \sum_t x^* \nabla_t + \eta \sum_t x^* |\nabla_t| + \frac{G_\infty \log n}{\eta}$$

Where we denote $|\nabla_t|$ for the vector that has in coordinate i the value $|\nabla_t(i)|$. Therefore

$$\sum_t f_t(x_t) - f_t(x^*) \leq \sum_t \nabla_t(x_t - x^*)$$

$$\leq \eta \sum_t |\nabla_t| x^* + \frac{G_\infty \log n}{\eta}$$

$$\leq \eta T G_\infty + \frac{G_\infty \log n}{\eta}$$

And the proof follows choosing $\eta = \sqrt{\frac{\log n}{T}}$

$\square$

Remark: As the algorithm is phrased, it needs to know T and G_∞ in advance (this is not a problem for the way we use online algorithms as a building block in approximate optimization). Standard techniques can be used so that the algorithm need not accept any input: the dependence on T can be removed by doubling the value of T as it is being exceeded. The dependence on G_∞ can be removed by using, at any point in the algorithm application, the largest G_∞ value encountered thus far.

D   Projections onto convex sets

Many of the algorithms for online convex optimization described in this chapter require to compute projections onto the underlying convex set. This correspond to the following computational problem: given a convex set $\mathcal{P} \subseteq \mathbb{R}^n$, and a point $y \in \mathbb{R}^n$, find the point in the convex set which is closest in Euclidian distance to the given vector. We denote the latter by $\Pi_{\mathcal{P}}[y]$.

This problem can be formulated as a convex program, and thus solved in polynomial time by interior point methods or the ellipsoid method. However, for many simple convex bodies which arise in practical applications (some of which will be detailed in following chapters), projections can be computed much more efficiently. For the n-dimensional unit sphere, cube and the simplex these projections can be computed combinatorially in $\tilde{O}(n)$ time, rendering the online algorithms much more efficient when applied to these convex bodies.

The unit sphere   The simplex projection is over the unit n-dimensional sphere, which we denote by $\mathbb{B}_n = \{x \in \mathbb{R}^n \ , \ \|x\|_2 \leq 1\}$. Given a vector $y \in \mathbb{R}^n$, it is easy to verify that it's projection is

$$\Pi_{\mathcal{P}}[y] = \begin{cases} y & \|y\| \leq 1 \\[2mm] \frac{y}{\|y\|} & o/w \end{cases}$$

The unit cube   Another body which is easy to project onto is the unit n-dimensional cube, which we denote by $\mathbb{C}_n^n = \{x \in \mathbb{R}^n \ , \ \|x\|_\infty \le 1\}$ (i.e. each coordinate is less than or equal to one). Given a vector $y \in \mathbb{R}^n$, it is easy to verify that it's projection is

$$\forall i \in [n] \ . \ \Pi_{\mathcal{P}}[y](i) = \begin{cases} y[i] & y(i) \in [-1, 1] \\ 1 & y(i) > 1 \\ -1 & y(i) < -1 \end{cases}$$

The Simplex   The first non-trivial projection we encounter is over the n-dimensional simplex. The simplex is the set of all n-dimensional distributions, and hence is particularly interesting in many real-world problems, portfolio management and haplotype frequency estimation just to name a few. Surprisingly, given an arbitrary vector in Euclidian space, the closest distribution can be found in near linear time. A procedure for computing such a projection is given in figure 6.

SIMPLEXPROJECT **(y).**
Suppose w.l.o.g that $y_1 \le y_2 ... \le y_n$ (otherwise sort indices of y).

- Let $a \in \mathbb{R}$ be the number such that $\sum_{i=1}^n \max\{y_i - a, 0\} = 1$. Set
  $\forall i \in [n] \ . \ x_i = \max\{y_i - a, 0\}$.

- **Return** x

Figure 6: A Procedure for projecting onto the Simplex

Lemma 14. SIMPLEXPROJECT (y) *is the projection of $y \in \mathbb{R}^n$ to the n-dimensional simplex, and can be computed in time $\tilde{O}(n)$.*

Proof. First, note that the number a computed in SIMPLEXPROJECT exists and is unique. This follows since the function $f(a) = \sum_{i=1}^n \max\{y_i - a, 0\}$ is continuous, monotone decreasing, and takes values in $[0, \infty)$.

Next, the vector returned $x = SimplexProject(y)$ is in the simplex. All its coordinates are positive by definition, and $\sum_{i=1}^n x_i = \sum_{i=1}^n \max\{y_i - a, 0\} = 1$.

To show that x is indeed the projection we need to prove that it is the optimum of the mathematical program

$$\min_{x \in S_n} \sum_{i=1}^n (y_i - x_i)^2$$

It suffices to show that x is a local optimum, since the program is convex. Let $c_i \triangleq y_i - x_i$. Then the values $\{c_i\}$ are decreasing and of the form

$$(c_1, ..., c_n) = (a, ..., a, y_k, ..., y_n)$$

An allowed local change is of the form $x_i' \leftarrow x_i - \varepsilon$ and $x_j' \leftarrow x_j + \varepsilon$ for $i < j$, since all coordinates larger than k have $x_k = 0$. This would cause a change in the objective of the form

$$\sum_{i=1}^d (y_i - x_i)^2 - (y_i - x_i')^2 = a^2 - (a + \varepsilon)^2 + c_j - (c_j - \varepsilon)^2 = -2(a - c_j)\varepsilon - 2\varepsilon^2 < 0$$

21

Hence would only reduce the objective. Therefore x is indeed the projection of y.

The procedure SIMPLEXPROJECT requires sorting n elements, and finding the value a, which is standard to implement in $O(n \log n) = \tilde{O}(n)$ time.

$\square$

E Examples of strictly convex mathematical programs

In this section we give some examples of problems which arise in practice and contain strictly convex constraints. The first example henceforth, and many others, appear in the excellent survey of [LVBL98].

E.1 Portfolio optimization with loss risk constraints

A classical portfolio problem described in [LVBL98] is to maximize the return of a portfolio over n assets under constraints which limit its risk. The underlying model assumes a gaussian distribution of the asset prices with known n-dimensional mean and covariance matrix.

The constraints bound the probability of the portfolio to achieve a certain return under the model. A feasibility version, of just checking whether a portfolio exists that attains certain risk with different mean-covariance parameters, can be written as the following mathematical program

$$p_j^\top x - \beta \cdot x^\top \Sigma_j\ x \geq \alpha \quad \forall j \in [m] \tag{11}$$
$$x \in \mathbb{S}_n$$

We refer the reader to [LVBL98] section 3.4 for more details.

If the underlying gaussian distributions are not degenerate, the covariance matrices Σ_j are positive definite. If the covariance matrices are degenerate - there is a linear dependance between two or more assets. In this case it is sufficient to consider a smaller portfolio with only one of the assets.

The non-degeneracy translates to a strictly positive constant $H > 0$ such that $\forall j \in [m]$. $\Sigma_j \succeq H \cdot I$. This is, of course, the smallest eigenvalue of the covariance matrices.

E.2 Computing the best CRP in hindsight with transaction costs normalization

In a popular model for portfolio management (see [Cov91, HSSW96]) the market is represented by a set of price relative vectors $r_1, ..., r_T \in \mathbb{R}_+^n$. These vectors represent the daily change in price for a set of n assets. A Constant Rebalanced Portfolio is an investment strategy that redistributes the wealth daily according to a fixed distribution $p \in \mathbb{S}_n$. A natural investment strategy computes the best CRP up to a certain trading day and invests according to this distribution in the upcoming day.

On this basic mathematical program many variants have been proposed. In [AH05], a logarithmic barrier function is added to the objective, which enables to prove theoretical bounds on the performance. Bertsimas [Ber06] suggested to add a quadratic term to the objective function so to take into account transaction costs. An example of a convex program to find the best CRP, subject

to transaction costs constraints is

$$\max \sum_{t=1}^{T} \log(p^\top r_t) + \sum_{i=1}^{n} \log(p^\top e_i) \tag{12}$$
$$\|p - \tilde{p}\|_2^2 \leq c$$
$$p \in \mathbb{S}_n$$

The objective function includes the logarithmic barrier of [AH05], the vectors $\{e_i\}$ are the standard basis unit vectors. The constraint enforces small distance to the current distribution $\tilde{p}$ to ensure low transaction costs.

The Hessian of the objective is

$$\sum_{t=1}^{T} \frac{1}{(p^\top r_t)^2} r_t r_t^\top + \sum_{i=1}^{n} \frac{1}{p_i^2} e_i e_i^\top \succeq I$$

The Hessian of the constraint is the identity matrix. Hence the constant H for Theorem 3 is one.

E.3   Maximum entropy distributions for with ℓ_2 regularization

The following mathematical program arises in problems concerning frequency estimation from a given sample. Examples include modelling of species distributions [DS06] and haplotype frequency estimation [HH06].

$$\min H(p) \tag{13}$$
$$\|A_i(p - \tilde{p})\|_2^2 \leq c \quad i \in [m]$$
$$p \in \mathbb{S}_n$$

Where $H : \mathbb{R}^n \mapsto \mathbb{R}$ is the negative of the entropy function, defined by $H(p) = \sum_{i=1}^{n} p_i \log p_i$. The Hessian of H is $\nabla^2 H(p) = diag(\frac{1}{p})$, i.e. the diagonal matrix with entries $\{\frac{1}{p_i}, i \in [n]\}$ on the diagonal. Hence $\nabla^2 H(p) \succeq I$.

The hessian of the i'th constraints is $A_i A_i^\top$. For applications with $\min_i A_i A_i^\top \succeq c \cdot I$, the constant H for Theorem 3 is $H = \min\{1, c\}$.

F   Proof of Corollary 4

proof of Corollary 4. Given mathematical program (1), we consider the following program

$$f_j(x) + \delta\|x\|_2^2 - \delta \leq 0 \quad \forall j \in [m] \tag{14}$$
$$x \in \mathbb{S}_n$$

This mathematical program has strictly convex constraints, as

$$\forall i \in [m] . \nabla^2(f_i(x) + \delta\|x\|_2^2 - \delta) = \nabla^2 f_i(x) + 2\delta I \succeq 2\delta I$$

Where the last inequality follows from our assumption that all constraints in (1) are convex and hence have positive semi-definite Hessian. Hence, to apply Theorem 3 we can use $H = 2\delta$. In addition, by the triangle inequality the gradients of the constraints of (14) satisfy

$$\|\nabla(f_i(x) + \delta\|x\|_2^2 - \delta)\|_2 \leq \|\nabla f_i(x)\|_2 + 2\delta \leq G + 2\delta = O(G)$$

Where G is the upper bound on the norm of the gradients of the constraints of (1). Therefore, Theorem 3 implies that a ε-approximate solution to (14) can be computed in $\tilde{O}(\frac{G^2}{\delta\varepsilon})$ iterations, each requiring a single gradient computation and additional $\tilde{O}(n)$ time.

Notice that if (1) is feasible, i.e there exists $x^* \in S_n$ such that $\min_{i\in[m]} f_i(x^*) \leq 0$, then so is (14) since the same x^* satisfies $\min_{i\in[m]} f_i(x^*) + \delta\|x\|_2^2 - \delta \leq \delta\|x\|_2^2 - \delta \leq 0$.

Given a ε-approximate solution to (14), denoted y, it satisfies

$$\forall j \in [m] \ . \ f_j(y) + \delta\|y\|_2^2 - \delta \leq \varepsilon \ \Rightarrow \ f_j(y) \leq -\delta\|y\|_2^2 + \delta + \varepsilon \leq \delta + \varepsilon$$

Hence y is also a $(\varepsilon + \delta)$-approximate solution to (1).

Choosing $\delta = \varepsilon$, we conclude that a 2ε-approximate solution to (1) can be computed in $\tilde{O}(\frac{G^2}{\varepsilon^2})$ iterations. $\qquad\square$